Optimizers Comparison in an Artificial Neural Network for Image Classification

Luis Martinez-Caballero
Institute of Control and Industrial Electronics
Warsaw University of Technology
Warsaw, Poland
luis.martinez@pw.edu.pl

***Abstract*—Artificial neural networks (ANN) development has been increasing in the last few years due to advancements in computational power. Moreover, their implementation is finding different applications for different environments, one of them is on image processing. This paper comprehensively describes how to build an ANN for image classification and is trained and validated in the publicly available dataset of Fashion MNIST, which contains 6000 samples of 10 different cloth items. Furthermore, three different optimizers are compared: Stochastic Gradient Descent with Momentum, Root Mean Square Propagation and Adaptive Momentum as one of the most widely-used optimizers.**

***Index Terms*—Artificial Intelligence, Feedforward Neural Network, Image Classification, Optimizer.**

I. Introduction

This document is a report for the course Statistic and Theory of AI and its application to engineering sciences.

II. ANN architecture

The ANN is comprised of a set of components that produce the given capabilities of the network, for the case of the simplest architecture of ANNs, that is Feedforward ANNs the possibilities range from regression, pattern recognition, function approximation and classification.

A. Neuron

The neuron is the fundamental part of an ANN, an scheme of a neuron is presented in Fig. 1. A given input x can have a set of n features passed to the neuron. Then $x = x_0, x_1, ..., x_n$ A single neuron has two parameters that are modified to improve the performance of the network: the weights and the bias. The weights are a set of numbers $w = w_0, w_1, ..., w_n$ and the bias is defined by the symbol b. It can be noted, that there are as many weights as input features, the first operation that is performed in a single neuron is the dot product between the input x and the weights b and at the end, adding a bias, this operation is given by (1).

$$z = w \cdot x + b \tag{1}$$

It is worth noting that this is a linear operation and, in order to produce more complex relations such as non-linear operation, an activation function is applied to the value of z.

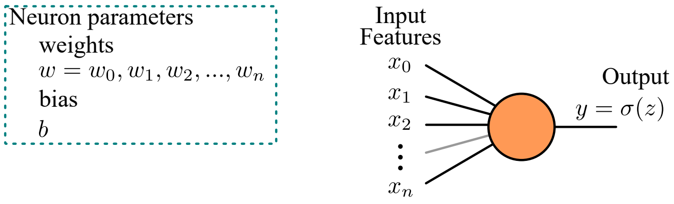


Fig. 1. Single neuron diagram.

B. Layers

A fully connected ANN is represented in Fig. 2. In this diagram there are several layers that comprise the ANN. First, we have the raw input to the ANN which is represented by x_0 and x_1. These inputs are passed to the first layer, also known as the input layer, it can be noted that every input is sent to each neuron of the input layer. Each neuron has its set of weights and biases, as previously discussed and in some cases an activation function is applied to the output.

The subsequent layers are called the hidden layers of the ANN, it can be noted that in each every layer the outputs of the previous layer are transferred to each neuron of next layer. This kind of connection describes a fully connected ANN which is the one used in this work. It is important to mention that there are different configurations where the layers are not fully connected.

This work uses a fully connected feedforward neural network of two layers. The first and second layer have 128 neurons each, while the output layer has 10 neurons, this is because in the dataset there are 10 different classes for the data to be classified.

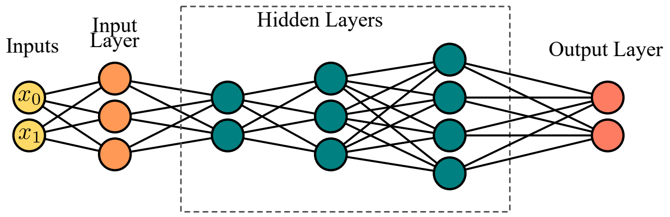


Fig. 2. Architecture of a fully connected ANN.

C. Activation Function

Different activation functions may be used depending on the application and the task to be performed. In this paper the Rectified Linear Unit (ReLU) and the Softmax functions will be discussed, these activation functions are set in different parts of the NN, in this case the ReLU is used for the input and the hidden layers, meanwhile the Softmax function is used for the output layer.

The ReLU function's output produces a zero value when its input is less or equal to zero and passes the input when it is greater than zero. This function is described by (2), and can be visualized in Fig. 3.

$$y = max(0, z) \tag{2}$$

It is important to highlight that the ReLU function is non-linear, and this property allows to generate more complex relations when adding more layers to the neural network, for example, when the task to perform is function approximation.

As the final goal in this study is to classify a set of images, the Softmax function is needed for the final layer of the NN. This activation function aims to get confidence scores, or in other words, a set of probabilities that will add to one, where the highest values correspond to the prediction of a particular class for a given input. The formula of the softmax activation function is given by (3).

$$y = \frac{e^{z_j}}{\Sigma_{l=1}^{L} e^{z_l}} \tag{3}$$

Index j represents the current output, and index l is used to iterate throughout each of the outputs of the last layer neurons. Suppose that we want to predict three classes; therefore, we have three neurons in the output layer, and the outputs of this layer are given by $Z = \{1.1, 2.3, 3.7\}$. If the Softmax activation is applied to the first output, 1.1, this results in:

$$y = \frac{e^{1.1}}{e^{1.1} + e^{2.3} + e^{3.7}} = 0.056 \tag{4}$$

Further applying the Softmax function to each output of the last layer, we obtain the vector $\sigma(Z) = \{0.056, 0.187, 0.757\}$ which represents the set of probability distributions for each class, where the last value is the highest and is the predicted class from a given input. Additionally, it is worth noting that the values add to one, as previously mentioned. One of the main drawbacks of using the exponential function is that it can lead to big values that can cause overflow. Therefore, one more step is added: subtraction of the maximum value. This operation will not change the probability distributions. Let's use the previous example where the outputs are given by $Z = \{1.1, 2.3, 3.7\}$, subtracting the maximum value will produce $Z' = \{-2.6, -1.4, 0.0\}$. Applying the softmax function to these values results in $\sigma(Z') = \{0.056, 0.187, 0.757\}$ which is equal to $\sigma(Z)$.

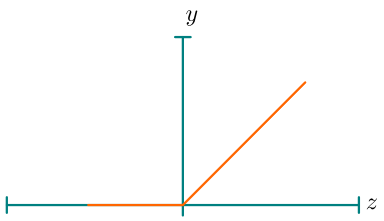


Fig. 3. ReLU activation function.

III. Dataset Description

The dataset used for this work is known as the fashion MNIST dataset [4]. This dataset is intended for image classification, and it has 6000 images to be used as training data for each class of clothing articles; ten different classes make a total of 60000 images of training data.

There are additional 1000 images for each class to be used as validation data. Each image is in grayscale format where each pixel is represented with a value from 0 to 255, and the size of each image is 28 x 28 pixels. Table I summarizes each class and its number representation. One of the images of the training data set is shown in Fig. 4

A. Scaling

As previously mentioned, the image is shown in grayscale; therefore each pixel of the images is represented with a value from 0 to 255. Neural networks are preferred to receive input data from the range 0 to 1 or -1 to 1. Some of the reasons why scaling is preferred are: better convergence, avoid vanishing or exploding gradients and efficient weight update, which in turn results in a better performance of the ANN. In this work the dataset is scaled to the range -1 to 1.

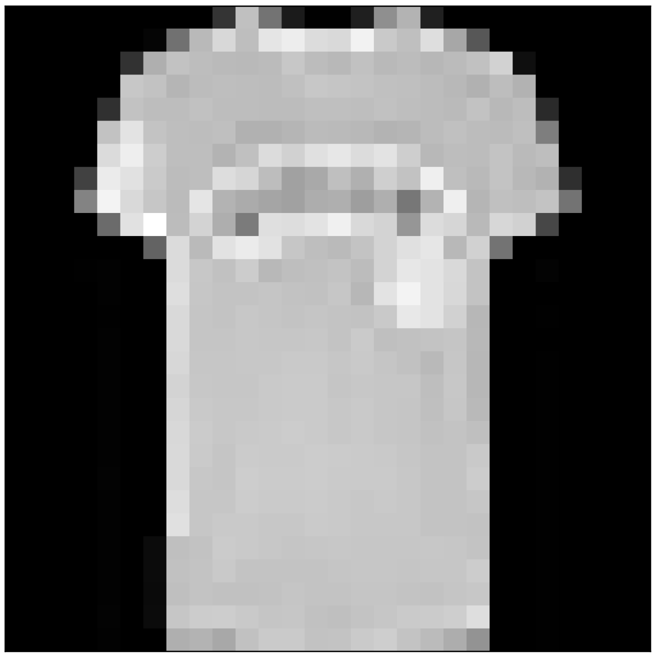


Fig. 4. Example of one of the figures taken from the data set, class 0 example.

TABLE I
LIST OF CLASSES OF THE DATASE

Number	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

B. *Shuffling*

Data shuffling is an additional and crucial step to take into consideration. The data is already sorted for each class; however, feeding the data to the model for training in this manner doesn't allow for good training. As the input is first only one class, the accuracy increases only for that class; then when a new class is fed into the ANN for training, a spike will occur for both loss and accuracy as the new class has never been used for training purposes.

The shuffling step is only performed in the training data, as the validation data is only used to test the final performance of the ANN without further update of the weights and biases.

IV. COMPARSION OF OPTIMIZERS FOR CLASSIFICATION

A. *Stochastic Gradient Descent with Momentum*

For this case we are going to change the decay value for the learning rate to different values in the SGD to get the best-performing model in terms of accuracy for training , to later test in validation test.

The results regarding the training using the SGD optimizer are shown in Fig. 5, four different decay values where used, and it can be noted that the best performance is for values in the middle of the list. It is worth mentioning that for the case of this optimizer for this application it is better to use larger values as a very small value causes a low accuracy and greater loss.

B. *Root Mean Square Propagation*

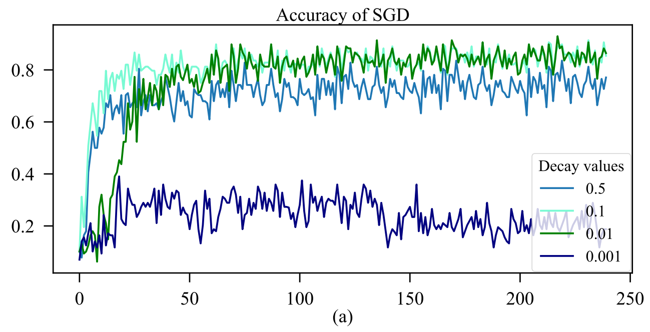
The Root Mean Square Propagation (RMSprop) optimizer has been tested. This optimizer calculates an adaptive learning rate per parameter, and is an adaptation of the SGD.

The RMSprop adds a mechanism similar to momentum together with a per parameter adaptive learning rate to produce smoother changed.

Different decay values were tested in order to get the best performance of the model that can be compared with the SGD results. The accuracy and loss data for different decay values is presented in Fig. 6

C. *Adaptive Momentum*

The Adaptive Momentum (Adam) optimizer is one of the most used algorithms for updating parameters in ANNs [1], [2].



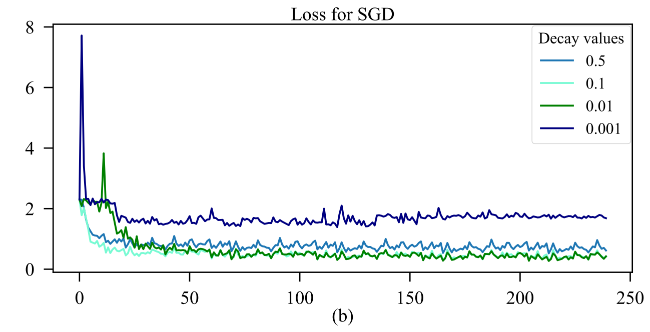
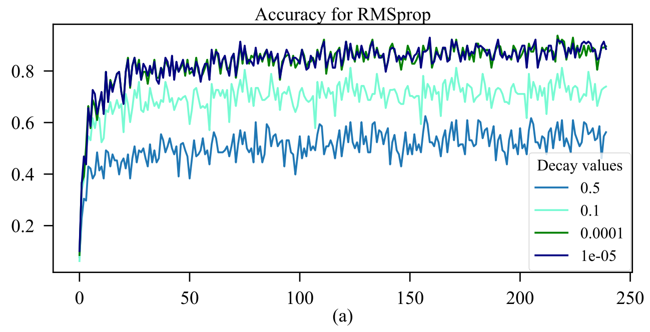


Fig. 5. Training results using the SGD optimizer. (a) Accuracy. (b) Loss.



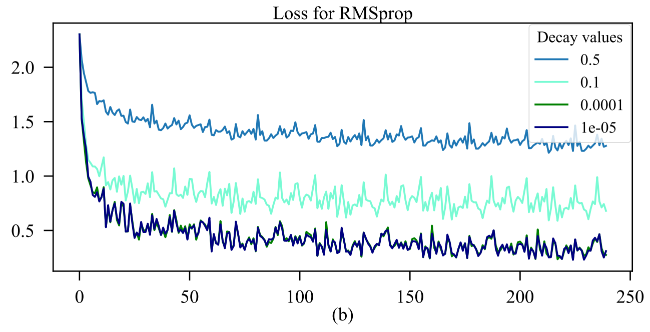


Fig. 6. Training results using the RMS propagation optimizer. (a) Accuracy. (b) Loss.

This optimizer is similar to the RMSProp, including the momentum as in the SGD optimizer. Similarly, an adaptive learning rate is applied per weight. Additionally, a correction mechanism is added to momentum and cache to compensate for the initial zero values. Results of different decay values are shown in Fig. 7, where it can be observed that similar to RMSprop, lower decay values provide better performance of

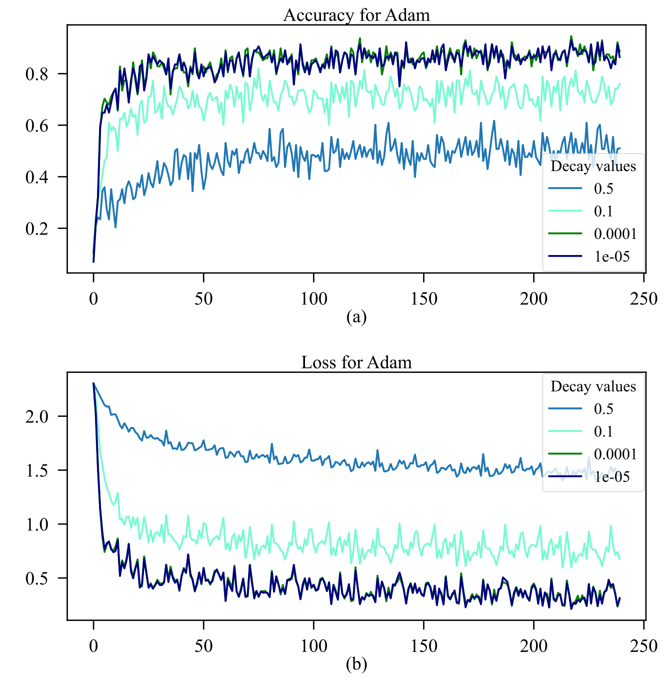


Fig. 7. Training results using the Adam optimizer. (a) Accuracy. (b) Loss.

the model.

D. *Comparison between optimizers*

The comparison between different optimizers is shown in Fig.8.

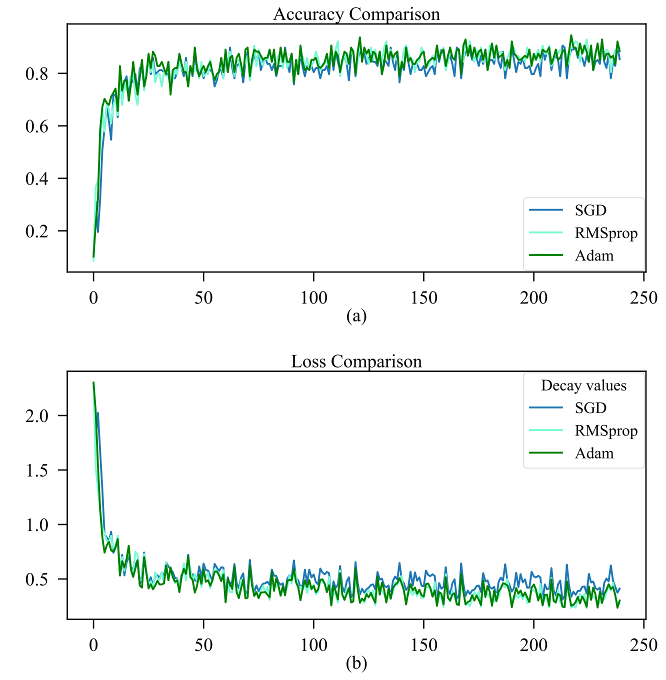


Fig. 8. Comparison of different optimizers during training. (a) Accuracy. (b) Loss.

V. Conclusions

Different optimizers where tested in the training of an image classification model. Results show that regardless of the optimizer used accuracy and loss results are similar once that the proper decay is set in each optimizer. However, it is important to highlight that for the SGD optimizer very low values of decay will produce a poor learning of the model. On the other hand, for the RMSprop and Adam optimizers it is prefered to start the training process with very small decay values around 1e-4 or 1e-5.

References

[1] Z. Chang, Y. Zhang and W. Chen, ”Effective Adam-Optimized LSTM Neural Network for Electricity Price Forecasting,” 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2018, pp. 245-248.

[2] R. Llugsi, S. E. Yacoubi, A. Fontaine and P. Lupera, ”Comparison between Adam, AdaMax and Adam W optimizers to implement a Weather Forecast based on Neural Networks for the Andean city of Quito,” 2021 IEEE Fifth Ecuador Technical Chapters Meeting (ETCM), Cuenca, Ecuador, 2021, pp. 1-6.

[3] G. Cong and O. Bhardwaj, ”A Hierarchical, Bulk-Synchronous Stochastic Gradient Descent Algorithm for Deep-Learning Applications on GPU Clusters,” 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, 2017, pp. 818-821.

[4] H. Xiao, K. Rasul and R. Vollgraf, “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms,” 2017 [arxiv].